REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YYYY)*** 14-04-2004	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From – To)* 22 May 2003 - 22-Mar-04

4. TITLE AND SUBTITLE

Statistical Characterization of MP3 Encoders for Steganalysis: 'CHAMP3'

5a. CONTRACT NUMBER FA8655-03-1-3A46

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dr. Andreas Westfeld

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Technische Universität Dresden (Technical University of Dresden in Germany)
Hans-Grundig-Str. 25
Room 320 B
Dresden D-01307
Germany

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Grant 03-3046

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. RETENTION PERIOD: A related paper is submitted to the ACM Multimedia and Security Workshop 2004. Please do not release this report to the public before October 2004.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Technische Universität Dresden (Technical University of Dresden in Germany) as follows: Grantee will investigate and characterize MP3 encoders and statistically analyze MP3 data to enable detection of embedded information in MP3 files and streaming data. This may also assist with identification of the origin of MP3 files (i.e. provide information regarding the type of MP3 encoder used to generate sample data). As detailed in the technical proposal, the research consists of three parts. The first task is to survey the discipline to identify the available MP3 encoders and generate a data pool for analysis. The second task is analysis of the encoders and statistical classification of the data they produce. The third task is to characterize the robustness of techniques developed by testing the approach on known and unknown reference data.

15. SUBJECT TERMS
EOARD, Steganography, Digital Watermarking, Statistical Methods

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** PAUL LOSIEWICZ, Ph. D. PBL
a. REPORT UNCLAS	**b. ABSTRACT** UNCLAS	**c. THIS PAGE** UNCLAS	UL		**19b. TELEPHONE NUMBER** *(Include area code)* +44 20 7514 4474

CHAMP3 Final Report[*]

Statistical Characterisation of MP3 Encoders for Steganalysis

Andreas Westfeld and Rainer Böhme
{westfeld,rainer.boehme}@inf.tu-dresden.de

Technische Universität Dresden
Institute for System Architecture
01062 Dresden, Germany

April 27, 2004

Abstract

This report describes a method to discriminate different ISO/MPEG 1 Audio Layer-3 (MP3) encoding programs by statistical characteristics of the compressed audio files. Encoders are classified according to 10 statistical features by a *Naïve Bayes Classifier*, which can be induced from a set of test data. A classification experiment led to a success rate of 96 % correct classifications. The authors discuss all features and refer to possible limitations of feature selection. In addition, empirical evidence showed that a pre-classification of MP3 encoders increases the reliability of detection methods for steganographic content. The proposed scheme reduced the error rate of an attack against `mp3stego` from about 75 % false positives to 15 % misses. Implications for the generalisability to other file formats, and contributions for related applications are also addressed.

Keywords: Steganalysis, MP3 Encoder Classification, Digital Forensics

[*]This project was supported by the Air Force Office of Scientific Research under the research grant number FA8655-03-1-3A46. The U. S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation there on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Air Force Office of Scientific Research, or the U. S. Government.

Contents

List of Figures

List of Tables

List of Abbreviations

ACM	1. Audio Compression Manager
	2. Association for Computing Machinery
API	Application Programming Interface
CBR	Constant Bit Rate
CD	Compact Disc
CHAMP3	Statistical Characterisation of MP3 Encoders (Project title)
CRC	Cyclic Redundancy Check
DSP	Digital Signal Processing
FFT	Fast Fourier Transformation
FLD	Fisher Linear Discriminant Analysis
GUI	Graphical User Interface
ID3	Identifier for MP3 files
IIS	(Fraunhofer) Institut Integrierte Schaltungen
IJG	Independent JPEG Group
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
JSP	Java Server Pages
kbps	Kilobits per Second
LGPL	Lesser General Public License
LSB	Least Significant Bit
MB	Megabyte
MDCT	Modified Discrete Cosine Transformation
MP3	ISO/MPEG 1 Audio Layer-3
MPEG	Moving Picture Experts Group
NBC	Naïve Bayes Classifier
PCM	Pulse Code Modulation
SCFSI	Scale Factor Selection Information
SCMS	Serial Copy Management System
SQAM	Sound Quality Assessment Material
SVM	Support Vector Machine
VBR	Variable Bit Rate

1 Introduction

The invention of the ISO/MPEG 1 Audio Layer-3 (MP3) audio compression algorithm [7, 14] is probably one of the most remarkable and far-reaching developments in the area of digital media processing. The MP3 format enables compression rates of about 1/10 of the size of uncompressed digital audio while degrading the audible quality only marginally. Together with the moderate complexity of the compression algorithm—software implementations of MP3 coders/decoders (codecs) with acceptable performance even on low budget home computers soon became available—the format simplified the interchange of music and resulted in worldwide popularity for its users and sleepless nights for the music industry. The popularity of the format fostered demand for encoding tools and opened a market for a variety of programs for different needs. Today we count hundreds of MP3 encoder front-ends based on several dozens of encoding engines ranging from proof of concept hacks to targeted products either tuned for high speed, or optimised to costly and flexible tools for professional studio requirements.

Given these facts, the MP3 format became an interesting carrier for steganographically hidden data. Steganography, which is somewhat linked with cryptography, aims to conceal the very existence of a confidential message by hiding it imperceptibly within other, less suspicious data [21]. MP3 is a promising carrier format for steganography in three ways. At first, the popularity of the format is an advantage, because exchanging common and widely used types of data is less conspicuous to an observer. For example, sharing an MP3 file over the Internet is a completely common task and doing so is a plausible form of communication. Second, MP3 files are typically between 2 and 4 megabytes (MB) in size and thus are larger than other commonly used formats (e. g., text documents or photographs as e-mail attachments). All forms of information hiding suffer from a small proportion of payload compared to the total amount of information, which is necessary to cover the message. So, larger file sizes simplify the handling of medium-sized payloads (e. g., a text message or a photograph). The inconveniences that come with splitting up messages over different carriers can be almost avoided for MP3 files. Third, the nature of the lossy MP3 compression itself makes it attractive for steganographic use. The information loss that is a concomitant of the encoding process creates a certain amount of unpredictability that can be exploited to carry hidden information securely.

Compared to the suitability of MP3 files for steganography, the amount of known steganographic tools for this format is still quite limited. `MP3Stego` [22] is based on the `8hz-mp3` encoder [1] and hides message bits in the parity of block length. Although this procedure is limited to a very low capacity, it is (under certain conditions, see below) detectable [26]. The attack is based on the analysis of statistical properties, i. e., the variance of block lengths in the MP3 stream. `Stego-Lame` [24] pursues another approach and embeds into uncompressed *Pulse Code Modulation* (PCM) audio data. The amount of information is so small and the embedding procedure so carefully selected, that a subsequent lossy MP3 compression does not erase the hidden information. This tool is still in an experimental stage. An appropriate attack is delivered in the same bundle. In addition to these publicly known stego-tools we expect some more being used in the wild. Although the complexity of MP3 compression exceeds those of typical steganographic tools (e. g., LSB image embedding), the availability of commented source codes for MP3 encoders facilitates the composition of derivates with steganographic extensions. Hence, advances in the detection of steganographic data in MP3 files are relevant.

The experience with the existing attack against `MP3Stego` shows that the detection procedure can distinguish MP3 files with and without steganographic content quite reliably if they are encoded with either `MP3Stego` or its underlying encoding engine [1]. However, files from other encoders tend to have similar statistical properties as steganograms from `MP3Stego` and thus are identified as false positives. Hence, the reliability of the detection algorithm heavily depends on the prior knowledge about the encoder of a particular file. While this situation might be sufficient for an academic attack or proof of concept, it is definitely not optimal for real world applications. In the fieldwork, we usually cannot expect any prior knowledge about the source of an arbitrary MP3 file. We therefore present a procedure to determine the encoder of MP3 files on the basis of statistical features that are typical for a certain implementation of the MP3 format specification. The insertion of a preclassification of MP3 encoders allows a steganalyst to run the appropriate detection algorithm for the determined encoder and thus dramatically decrease the amount of false positives. Thus it is believed that statistical classification of MP3 encoders can increase the reliability of detection procedures.

The rest of this report is organised as follows. In the next section we briefly review the relevant particularities of the MP3 format that are analysed for the extraction of statistical features. Section 3 sums up the project realisation and describes the main steps to follow the presented procedures. The features themselves are explained in Section 4. Experimental results that back the performance of the proposed scheme are presented in Section 5, before we discuss further applications and possible generalisations to other file formats in Section 6.

2 Analysis of MP3 Specification

The purpose of this section is not to repeat the architecture and specification of MP3 compression [2, 14], but to give a brief overview of those principles that are relevant as features for our proposed statistical classification. Hence, we focus on the latitudes that are left in the ISO specification, which leave space for different implementations. It is the vaguely defined particularities that finally lead to different output streams for the same input data.

2.1 Principles of MP3 Compression

The developers of MP3 audio compression included several techniques to maximise the relationship between perceived audio quality and storage volume. In contrast to previous schemes, they designed a two-track approach. On the *first track*, the audio information is split up into 32 equally spaced frequency subbands. These components are separately mapped into the time domain with a *Modified Discrete Cosine Transformation* (MDCT). The following quantisation step reduces the precision of the MDCT coefficients. As a last step, a lossless entropy encoding of the quantised coefficients leads to the compact representation of MP3 audio data. The *second track* is very important for the performance of MP3 encoding, because it is used as a control track. Also starting from the PCM input data, a 1024-point Fourier transformation is used to fit the local frequency spectrum as input to a psycho-acoustic model. This model emulates the particularities of human auditory perception and derives appropriate masking functions for the input signal. The model is used to control the choice of block types and quantisation factors in the first track. Hence, this two-track approach adaptively finds an optimal trade-off between data reduction and audible degradation for a given input signal.

Regarding the underlying data format, an MP3 stream consists of a series of *frames*. Synchronisation tags separate frames from other information sharing the same transmission or storage stream (e. g., video frames). For a given bit rate, all MP3 frames have a fixed compressed size and represent a fixed amount of 1152 PCM samples. Usually, an MP3 frame contains 32 bits of header information, an optional 16 bit *Cyclic Redundancy Check* (CRC) checksum, and two *granules* of compressed audio data. Each granule can be subdivided into one (mono) or two (stereo) *blocks*. Since the actual block size depends on the amount of information that is required to describe the input signal, it may vary between frames. To match the floating block sizes with the fixed frame sizes without wasting bandwidth, the MP3 standard introduces a so-called *reservoir* mechanism. Frames that do not use their full capacity are filled up (partly) with block data of subsequent frames. This method assures that local highly dynamic sections in the input stream can be stored with over-average precision, while less demanding sections allocate under-average space. However, the extent of reservoir usage is limited in order to decrease the interdependencies between more distant frames and to facilitate resynchronisation in the middle of a stream.

2.2 Level of Analysis and Related Work

In order to perform a statistical characterisation of MP3 encoders we have to find differences in the encoding process. These differences may have multiple causes. At the first glance, all loosely defined parameters in the specification are subject to different interpretations. However, the standard precisely describes a large set of critical parameters including the exact coefficients for the filter bank and threshold values for the psycho-acoustic model. Nevertheless, some implementations seem to vary or fine tune these parameters. In addition, performance evaluations may have led to sloppy implementations of the standard, such as shortcuts in the inner quantisation loop or the choice of non-optimal Huffman tables. Also, a number of parameter defaults for meta information are up to the implementor (e. g., the *Serial Copy Management System* (SCMS) flags, also known as *protection bit* [11]). All these variations together cause particular features in the output stream that are indications of a specific encoder and therefore are subject to a detailed analysis.

Table 1: Structure of ISO/MPEG 1 Audio Layer-3 encoding process

	Transformation Layer	Modelling Layer	Bit Stream Layer
Functionality			
	– Filter bank	– Quantisation loop	– Auxiliary data
	– MDCT transform	– Model decisions	– Frame header bits
	– FFT transform	– Table selection	– Checksums
			– Stream formatting
Points for analysis			
	– Frequency range	– Size control	– Surface information
	– Filter noise	– Model decisions	– SCMS protection bit
	– Audible artefacts	– Capability usage	– SCMS original bit

To structure the occurrences of implementation specific particularities in the MP3 encoding process, we will subdivide the process into three layers as shown in Table 1. The *transformation layer* includes all “passive” operations that directly affect the audio data, namely the filter bank, and the MDCT and *Fast Fourier* (FFT) time to frequency transformations, respectively. In this layer, variations in the filter coefficients or in the precision of the floating point operations may cause measurable features such as typical frequency ranges or additional noise components.

We define all "active" components of the compression algorithm as part of the *modelling layer*. These sub-processes are less close to the underlying audio data and mainly perform the trade-off between size and quality of the compressed data. In this layer, encoder differences basically occur in three ways:

1. Calculation of size control quantities, e. g., whether net or gross file sizes are used as reference for the bit rate control.

2. Model decisions: Different threshold values lead to different marginal distributions of control parameters over the data stream.

3. Capability usage: Some encoders do not support all compression modes specified in the MP3 standard.

The uppermost layer, which we call *bit stream layer*, handles the formatting of already compressed MP3 frames into a valid bit stream. These operations include the composition of frame headers, the optional calculation of CRC checksums for error detection, and the insertion of meta data. For instance, quasi-standardised ID3 tags [15] contain information about the names of artists, interprets, and publishers of audio files. Optional VBR (*variable bit rate*) headers store additional data evaluated by some MP3 players to display valid progress bars and enable efficient skipping within MP3 files with variable bit rate.[1] The existence of a certain kind of meta information and its default values may be used as indicator for the encoding program.

EncSpot [5], the only tool for MP3 encoder detection we know, relies on the deterministic surface parameters of the bit stream layer. As these parameters are easily accessible, it is also simple to erase or change their values and therefore trick this kind of encoder detection. Therefore we decided to use statistical features related with deeper structures of the encoder and thus are more difficult to manipulate. Our initial experiments with parameters of the transformation layer showed that those tend to be dependent on the type of audio data actually encoded. For example, it is impossible to measure encoder characteristics, such as the upper frequency bound, if the encoded audio material does not use the full range. Also, artefacts occur at typical envelopes or frequency changes that do not appear similarly in all kinds of music. Hence, we decided to focus our level of analysis to the modelling layer, which promises to deliver the most robust features in terms of source data independency and difficulty of manipulation.

[1]As MP3 has been specified for *constant bit rates* (CBR) the majority of MP3 files are encoded as CBR with one of the predefined rates. However, some encoding programs optionally encode each frame with a different bit rate (out of the predefines rates), thus enabling *variable bit rate* (VBR) streams with MP3.

2.3 Scope and Terminology

To precisely describe the nature of used features we introduce some formal notations. We denote a medium m as m_0 for the source (i. e., uncompressed) representation and as $m_i = e_i(m_0)$ if it is encoded with encoding program e_i. e_i is element of the set of n encoders $E = \{e_1, e_2, \ldots, e_n\}$. We write the set of all files encoded using e_i as $M_i = e_i(M_0)$, where M_0 is the set of all uncompressed source media.

The function $f(m)$ extracts a symbolic feature x from m. The vector of k different features

$$\mathbf{x} = \mathbf{f}(m) = (f_1(m), f_2(m), \ldots, f_k(m))$$

is called feature vector. The components of the feature vector $\mathbf{x}$ are selected to be as similar as possible for different media $m \in M_i$ encoded with the same encoder e_i, and also as dissimilar as possible for all encoded media $\overline{m} \in \{e_j(m_0)|j \neq i\}$ that are derived from m_0 by encoding it with other encoders. Therefore the information about the characteristics of the encoding program is consolidated in the value of $\mathbf{x}$.

Classifiers are algorithms which automatically classify an object, i. e., assign it according to its features to one of several predefined classes. As the literature contains multiple options, the choice of a specific algorithm for our purpose was determined by the conditions given in our application. *Fisher Linear Discriminant* (FLD) methods and *Support Vector Machines* (SVM) have already been successfully applied for steganalysis [18, 8]. These methods perform well for numeric (i. e., continuous) features, but are less suitable for symbolic features. Hence, we chose to apply a classifier which is based on Bayesian logic [17]. As we will show in Section 5, we get notable results with the simple *Naïve Bayes Classifier* (NBC) [4].[2]

We use a classifier c to establish the relation between a specific instantiation of $\mathbf{x} = \mathbf{f}(m_i)$ and the encoding program e_i that was used to create m_i. If we do not have any knowledge about the encoder, we can only derive probabilistic evidence about this assignment. For a given medium m a classifier tries to compute the conditional probabilities

$$P(e_i|\mathbf{f}(m)) = P(e_i|x_1 = f_1(m), x_2 = f_2(m), \ldots, x_k = f_k(m)),$$

with $1 \leq i \leq n$, and then selects the most probable encoder e_i, so that

$$P(e_i|\mathbf{f}(m)) > P(e_j|\mathbf{f}(m)),\ \forall\ e_j \in E \backslash \{e_i\},\ i = c(\mathbf{f}(m)).$$

[2]These results are coherent with the findings from a comprehensive evaluation of different classifiers: Compared to a set of complex classification models, the simple NBC performed equal or superior for many realistic decision problems [16].

The classifier's performance depends on its parameterisation, which can be induced from data. Therefore we assemble a training set

$$T = \{(i, e_i(m)) | 1 \leq i \leq n \land m \in M_0\}.$$

Each element of T contains a compressed representation of medium m and a reference to the known encoding program. We note a classifier trained with the training set T as c_T. The encoder prediction of a specific instantiation of $\mathbf{x}$, and of an underlying medium m will be denoted as $c_T(\mathbf{x})$ and $c_T(\mathbf{f}(m))$, respectively. To evaluate the quality of the classification, we regard the proportion p of correctly classified cases when the classifier is run on elements of a test set S, which is composed similarly to the training set T:

$$p(c, S) = \frac{|\{(i, m_i) \in S | i = c(\mathbf{f}(m_i))\}|}{|S|}$$

As a weak form of reliability evaluation, the same training set T can be reclassified, thus $c_T(\mathbf{f}(m_i))$ with $(i, m_i) \in T$. A somewhat stronger measure can be achieved for disjoint test and training sets, so that $S \cap T = \emptyset$.

3 Procedure

The CHAMP3 Project was accomplished between June 2003 and March 2004 at Technische Universität Dresden, Germany. It has been organised into three main tasks. First, we accomplished preparatory tasks such as the development of analytical tools for MP3 files (cf. Section 3.2) and the set-up of a test database of MP3 files from different encoders. The encoder selection process and the obstacles we faced are described in Section 3.1. Second, analytical research was conducted to find distinguishing features between different encoders. This task is difficult to describe because of its iterative and explorative nature. It was mainly driven by systematic comparisons of the most promising parameters of MP3 files. The third task included further tests, the implementation of a prototype, and final documentation. The resulting sources and binaries are attached to this report and a brief tutorial of how to use the provided software is given in Appendix A.2.

3.1 Set-up of Test Database

To arrange a training set for our experiments, we had to collect the most important MP3 encoders. The search was oriented to well-known names, such as Fraunhofer or Xing, as well as to general advices from bulletin boards on the Internet. To complete our collection, we used the file names of archives as search terms in Google to get references to similar content, which turned out to contain further encoders.

We installed and examined several dozen MP3 encoders; however, we realised that most of them rely on the same back-end and simply differ in their user interfaces. The back-ends we found are mainly `l3enc` by Fraunhofer, `lame`, and `bladeenc`. While `bladeenc` and `l3enc` essentially appeared in one version, `lame` came in several builds from varying compilers and with different file names.

Some MP3 encoders provide the user with a large number of adjustable parameters. For instance, `lame` leaves dozens of independent choices for noise shaping, psycho-acoustic algorithms, filters, and bit rate (constant, average, and variable), thus offering several thousand possible combinations. Moreover, open-source tools in particular are frequently updated and new releases can have very minor changes. Since there are hardly any parameters supported by all MP3 encoders, we created the MP3 files for our database using the respective default parameters. The bit rate is the only parameter we varied. We used the bit rates 112 kbps, 128 kbps, and 192 kbps, which we believe to be the most commonly used ones. We back this assumption with evidence from an online user survey at `http://www.mp3-tech.org/`, and by counting several collections of MP3 files. The results are documented in Table 2. With some exceptions (`l3enc`, `mp3comp`, `shine`), the selected rates are supported by all other encoders.

The ideal audio content for our database would be a representative selection of music, either drawn from the set of commercially available recordings or sam-

Table 2: Share of bit rates of MP3 files

Bit rate	Estimated proportion	
	Poll on MP3tech.org (February 2003)	MP3 collections (about 20 K files)
320 kbps	8.0 %	0.2 %
224–256 kbps	7.2 %	0.4 %
160–192 kbps	27.0 %	12.9 %
128 kbps	22.6 %	69.5 %
112 kbps	1.0 %	4.9 %
64–96 kbps	3.0 %	9.3 %
up to 56 kbps	7.1 %	2.4 %
other	24.1 %	0.4 %
	100 %	100 %

pled from Peer-to-Peer (P2P) file sharing systems. However, we tried to find a compromise between an extensive database and a limited set-up time. So we had to limit the search in some respects according to our resources. For instance, we did not examine any hardware MP3 encoder.

It is desirable to have an automated set-up process for the database so that additional audio sources, encoders, or encoder settings can later be added with minor effort. This is a challenging task, because the encoders run on different platforms (Linux, DOS, MacOS, and Windows). For the test database, most encoders were called from shell scripts under Linux. A large part of the Windows and DOS software even runs under Linux with emulation software, such as `wine` [27] and `dosemu` [3]. To assure that these rather atypical application environments do not influence the resulting MP3 files, we compared the binary files with those from native platforms. Nonetheless, the remaining tools not working with one of the available emulations had to be run manually on the respective original platforms.

Windows encoders use so-called *Audio Compression Manager* (ACM) modules. These ACM libraries are stored in the Windows system directory. If Windows encoders use different ACM modules with identical file names, they mutually overwrite each other. Be aware that an encoder installed under Windows sometimes produces MP3 files with different properties after another encoder has been installed. Hence, it is advisable to install each encoder in its own Windows installation (e. g., under `VMware` [25]). Furthermore, many Windows encoders we found on the Internet are silently bundled with spyware.[3]

3.2 Tool Development

The *R Project for Statistical Computing* [23, 12] is a powerful language for data analysis and graphics, which comes with a wide variety of specific statistical functions. As it is released as free and open source software, it is highly extensible. We chose this technology as environment for the analytical parts of CHAMP3. However, we had to develop additional import functions to access statistical features of MP3 files directly. Our implementation of the interface library is based on `mpglib` [9], which is released under the *Lesser General Public License* (LGPL) [6]. The new function `load.mp3` returns an `mp3info` structure representing a given input MP3 file.

According to the prior explanation of the MP3 file format structure, we designed a data structure to map MP3 files into R objects. It is a convention that R functions store complex data structures in so-called `list` objects. The members of a list are accessible either by name—like associative arrays—or by number. The `mp3info` structure contains up to four nested lists, one for each data parsing level:

- `global`: a list of file specific information
- `frames`: a data frame object holding the header parameters for each frame
- `blocks`: a data frame object holding information for every block; for stereo files, two adjacent entries represent one granule
- `coeff`: a data frame object holding the 576 quantised MDCT coefficients for each block

The exact structure of the `mp3info` object, including all relevant attributes, is depicted in Figure 1. As the R representation of MP3 objects allocates a multiple of the file length of an MP3 file in memory, the `load.mp3` function is highly configurable. This enables the user to extract only the information which he or she actually needs for a certain analytical task. For example, since none of the features we use for the classification requires the exact coefficient values (as already mentioned, we do not use characteristics of the transformation layer), we do not need to import the coefficient data at all.

[3]Spyware, sometimes called *spybots* or *tracking software*, is a technology to secretly gather information about the user and relay it to advertisers or other interested parties.

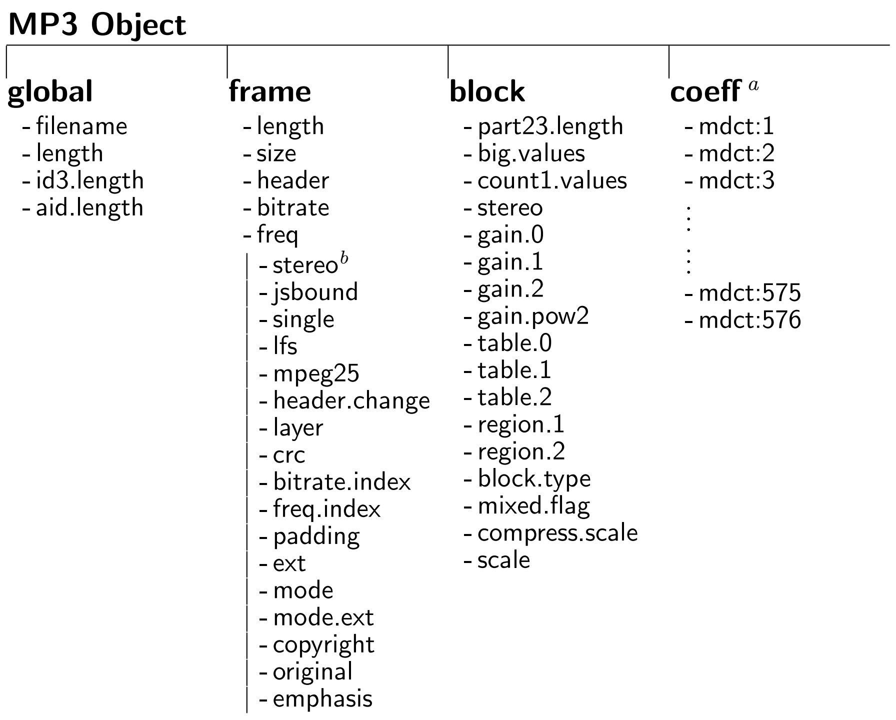


Figure 1: Structure of MP3 file as R object

[a]`load.mp3()` returns the following flags only if `expand.flags=TRUE` is specified.

[b]The `coeff` data frame is only included if the parameter `coeff` is explicitly set.

4 Description of Features

As a result of iterative comparisons and analyses of MP3 encoder differences, we discovered a set of 10 features in the *modelling layer*. For a structured presentation, the features are assigned to categories, which will be discussed separately in the following subsections.

4.1 Calculation of Size Control Quantities

Distinct encoders seem to differ in the way the target bit rate is calculated, as we discovered measurable differences in the effective bit rate. According to the MP3 standard, each block can be encoded with one of 14 predefined bit rates.[4] However, because of the difficulty to reach an exact compressed size, these act just as guiding numbers. Some encoders treat these rates as an upper limit, others as an average. Also, the encoders differ in the scope of frames that are evaluated as control parameters for the compression loop. If broader scopes are used, or fixed headers at the beginning of MP3 files are also reflected in the quantisation loop, then the effective bit rate varies with the file length and converges to a target value with an increasing number of frames.

4.1.1 Feature 1: Effective Bit Rate Ratio

These phenomena are depicted in Figure 2 for four selected encoders on the basis of files with a nominal bit rate of 128 kbps. The curves are drawn according to a least square estimate with a linear and a hyperbolic term over measured data points.[5] The effective bit rates β_{eff} of `8hz-mp3` and `mp3comp` depend on the number of frames r, while there is no influence for files encoded with `lame` or `fhgprod`. We calculate the effective bit rate as

$$\beta_{\text{eff}} = \frac{[(\text{filesize}) - (\text{junkbytes}) - (\text{meta information})] \cdot 8 \cdot \varphi}{1152 \cdot r},$$

with $\varphi = 44.1\,\text{kHz}$ as sampling frequency. Even for large files we observe a measurable difference in the marginal β_{eff} between all four encoders. To derive a bit rate independent feature from this observation, we calculate a criteria ϱ_1 as ratio between the effective bit rate β_{eff} and the nominal bit rate β_{nom}:

$$\varrho_1 = \frac{\beta_{\text{eff}}}{\beta_{\text{nom}}}, \quad \text{with} \quad \beta_{\text{nom}} = \frac{1}{r}\sum_{i=1}^{r} \beta_{\text{nom}}^{(i)},$$

where $\beta_{\text{nom}}^{(i)}$ is the nominal bit rate given in the header of the i-th frame. To map

[4]Bit rates for Layer-3 in kbps: 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320

[5]R^2 values range between 0.83 and 0.97.

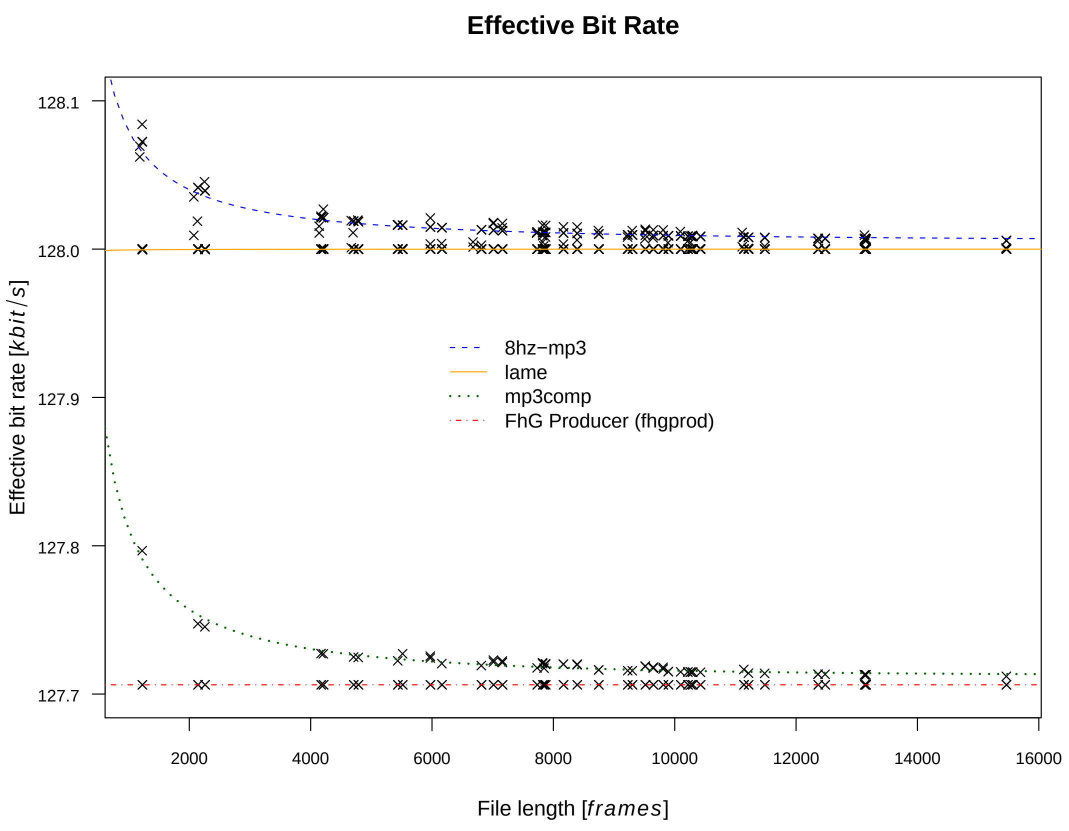


Figure 2: Relation between effective bit rate and file length for selected encoders

this ratio to a symbolic feature x_1, we define the extraction function f_1 as follows:

$$x_1 = f_1(m) = \begin{cases} 0 & \text{for} \quad \varrho_1 < 1 - 1\cdot 10^{-4} \\ 1 & \text{for} \quad 1 - 1\cdot 10^{-4} \le \varrho_1 \le 1 \\ 2 & \text{for} \quad 1 < \varrho_1 \le 1 + 5\cdot 10^{-6} \\ 3 & \text{else.} \end{cases}$$

The number of levels and the exact boundaries for this feature, as well as for the following ones, are determined by an iterative process of comparing a set of test audio files. We report the functions which lead to the best experimental results, even though further optimisation is still possible.

4.1.2 Feature 2: Granule Size Balance

In Section 2.1, we mentioned that an MP3 stream consists of a sequence of *frames*. Again, two *granules* share a frame of fixed size. The quantisation loop adjusts the size of the granules separately according to two criteria:

1. Size: The granule must fit into the available space.
2. Quality: Signal noise shall remain imperceptible.

For some encoders, e. g., `shine`, we observed a slight bias for quality over size. As the 'hard' space limit counts on both granules together, the first granules $g_1^{(i)}$ of all frames ($1 \leq i \leq r$) tend to get bigger than the second ones $g_2^{(i)}$. Hence, we measure the proportion of frames in the file where the length of the first granule $\text{len}(g_1)$ dominates the second one $\text{len}(g_2)$:

$$\varrho_2 = \frac{1}{r} \sum_{i=1}^{r} G(i), \qquad \text{with}$$

$$G(x) = \begin{cases} 1 & \text{for} \quad \text{len}(g_1^{(x)}) > \text{len}(g_2^{(x)}) \\ 0 & \text{else.} \end{cases}$$

Again, we define a mapping function, now for feature x_2:

$$f_2(m) = \begin{cases} 0 & \text{for} & & & \varrho_2 & < & 0.50 \\ 1 & \text{for} & 0.50 & \leq & \varrho_2 & < & 0.55 \\ 2 & \text{for} & 0.55 & \leq & \varrho_2 & < & 0.70 \\ 3 & \text{else.} \end{cases}$$

4.1.3 Feature 3: Reservoir Usage Ramp

The next feature makes use of characteristics of the reservoir mechanism. We found that the abruptness of the rise in reservoir usage between silent and dynamic parts in the audio stream differs between some encoders. Other encoders even do not use the reservoir at all. As the vast majority of audio files start with a tiny silence, we derive the feature x_3 from the amount of bytes shared between the first and the second frame $v_{(1,2)}$:

$$f_3(m) = \begin{cases} 0 & \text{for} & v_{(i,i+1)} = 0 & \forall\, i\colon 1 \leq i < r \\ 1 & \text{for} & v_{(1,2)} > 300 \\ 2 & \text{else.} \end{cases}$$

The function $f_3(m)$ is zero if the reservoir is not used in the whole file. The values 1 and 2 identify hard and soft reservoir usage, respectively.

4.1.4 Feature 4: Entropy of Big Values

The last feature in this category is less theoretically based and our evaluations show that it has little impact on the classification result, except for a better separation between two versions of the Xing encoder, namely `xing98` and `xing3`. However, we report it for the sake of completeness. We observed that `xing3` uses

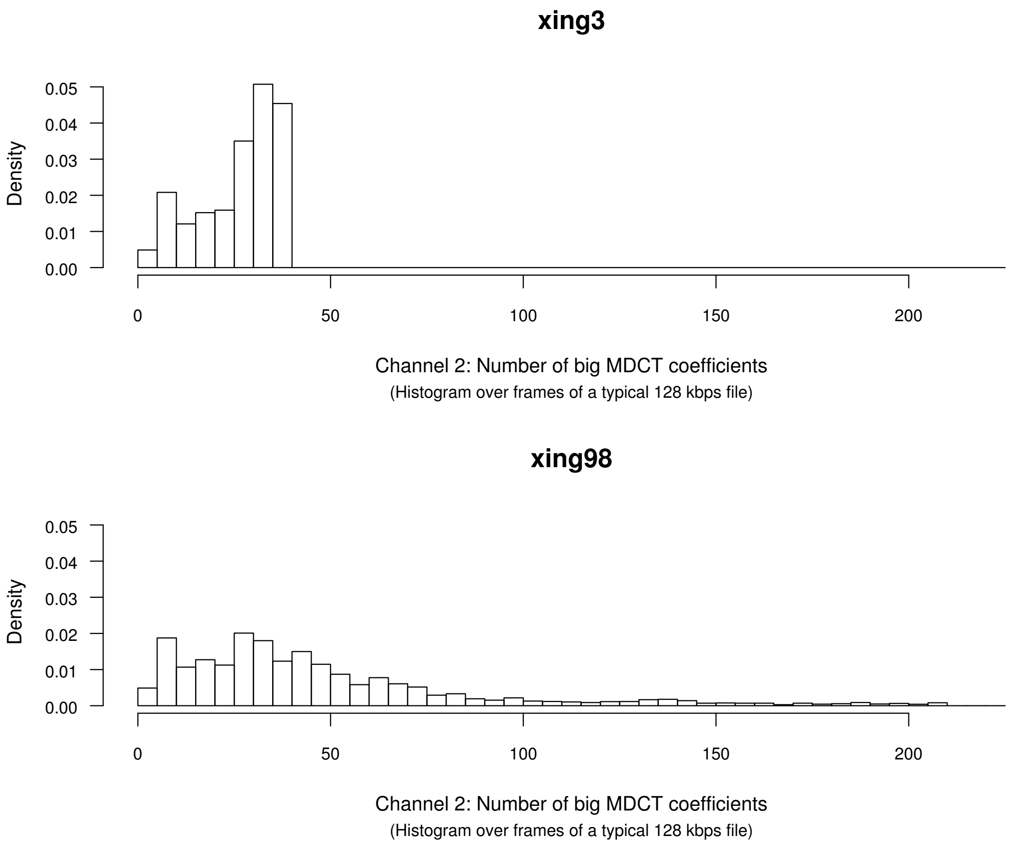


Figure 3: Comparison of size control in stereo files encoded with `xing3` and `xing98`

a different size control mechanism for the second block of every granule of stereo files. The differences are clearly visible in the histogram of lengths of *big value* MDCT coefficients (see Figure 3). Following the ISO/MPEG 1 Audio Layer-3 terminology [14], *big values* are the partition of spectral coefficients with absolute values greater than 1. This partition holds the most energy of the transformed audio signal and thus the average number of big values is a valid indicator for the extent of size reduction in the quantisation loop. To derive a continuous feature from the different spread of histogram values in the stereo channel, we measure the entropy from the histogram with the approximation given in [19]:

$$H \approx -\sum_{j=1}^{d_{\max}} d_j \log d_j + \log \Delta x,$$

with d_j denoting the density of occurrences in the j-th bin and Δx as bin size. Since Δx is constant for all encoders, we use a simplified function to calculate feature x_4:

$$f_4(m) = -\sum_{j=1}^{60} d_j \log d_j$$

Note that in contrast to previous features, $f_4(m)$ is a continuous feature that is modelled by the classifier as a normal distributed random variable with mean $\mu_i^{(4)}$ and standard deviation $\sigma_i^{(4)}$ for the i-th encoder e_i. However, as this feature evaluates the characteristics of the second channel in stereo data, it is not applicable to mono files; hence, we cannot discriminate between `xing3` and `xing98` for mono files.

4.2 Model Decision

The psycho-acoustic model is a second source for distinguishing features. Differences in the computation of control parameters or modifications in the choice of threshold values lead to typical marginal distributions of measurable parameters.

4.2.1 Feature 5: Preflag Ratio

The binary value *preflag* causes an additional amplification of high frequencies and is individually set for each compressed block b_i ($1 \leq i \leq q$, with q as number of blocks in a file). Concerning the treatment of this parameter, the ISO/MPEG 1 Audio Layer-3 standard explicitly leaves latitude:

> "The condition to switch on the preemphasis is up to the implementation." [14, p. 110]

To derive an operable feature we calculate the proportion of blocks with preflag set

$$\varrho_5 = \frac{1}{q} \sum_{i=1}^{q} \text{preflag}(b_i)$$

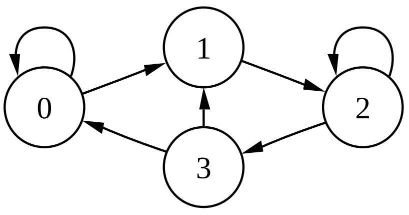
and map it into disjoint regions for the symbolic feature x_5:[6]

$$f_5(m) = \begin{cases} 0 & \text{for} & & & \varrho_5 & = & 0.00 \\ 1 & \text{for} & 0.00 & < & \varrho_5 & \leq & 0.01 \\ 2 & \text{for} & 0.01 & < & \varrho_5 & \leq & 0.05 \\ 3 & \text{for} & 0.05 & < & \varrho_5 & \leq & 0.10 \\ 4 & \text{for} & 0.10 & < & \varrho_5 & \leq & 0.21 \\ 5 & \text{for} & 0.21 & < & \varrho_5 & \leq & 0.35 \\ 6 & \text{for} & 0.35 & < & \varrho_5 & \leq & 0.62 \\ 7 & \text{for} & 0.62 & < & \varrho_5 & \leq & 0.77 \\ 8 & \text{else.} \end{cases}$$

[6]Our experiments show that the symbolic interpretation of x_5 leads to better classification results than a treatment as continuous feature with assumed normal distribution.

4.2.2 Feature 6: Block Type Transitions

The MP3 audio format offers different block types, which allow an optimal trade-off for sections requiring higher time resolution at the cost of frequency resolution and vice versa. The majority of blocks are encoded with block type 0, the *long block* with lower time and higher frequency resolution. Block type 2 defines a *short block*, which offers less coefficients to be stored for three different points in time. Two more block types are specified to perform smooth shifts between the above mentioned types. Hence, the standard defines a graph of valid block transitions between two adjacent blocks b_i and b_{i+1}, as shown in Figure 4.



$$\begin{tikzcd}
 & 1 \arrow[dr] & \\
0 \arrow[loop above] \arrow[ur] & & 2 \arrow[loop above] \arrow[dl] \\
 & 3 \arrow[ul] \arrow[uu] &
\end{tikzcd}$$

Figure 4: Valid MP3 block type transitions

An evaluation of block type transitions of MP3 files from different encoders uncovers two interesting details: First, some encoders (`shine`, all `xing*`) do not use short blocks at all and thus always encode with block type 0. Second, other encoders (`lame`, `gogo`, and `plugger`) include specific “illegal” transitions, mainly at the beginning of a file. As these transitions are rarely observable from other encoders, they identify the encoder reliably. Hence, we construct the extraction function for feature x_6 as follows:[7]

$$f_6(m) = \begin{cases} 0 \quad \text{for} \quad \text{type}(b_i) = 0 \quad \forall\, i\colon 1 \leq i \leq q & (\texttt{shine}, \texttt{xing*}) \\ 1 \quad \text{for} \quad \text{type}(b_1) = 0 \quad \wedge \quad \text{type}(b_2) = 2 & (\texttt{lame}) \\ 2 \quad \text{for} \quad \text{type}(b_1) = 2 \quad \wedge \quad \text{type}(b_2) = 3 & (\texttt{gogo}) \\ 3 \quad \text{for} \quad |\{b_i|\text{type}(b_i) = 2\}| = |\{b_i|\text{type}(b_i) = 3\}| = 1 & (\texttt{plugger}) \\ 4 \quad \text{else.} & \end{cases}$$

We have no other explication for these strange transitions than assuming that they are intended to leave a kind of encoder fingerprint in the output data. It is up to a deeper analysis of these particularities in the source code to reveal further evidence.

[7]For simplicity we give the relations for mono files. Stereo files work similar if blocks are evaluated in pairs. The given definition is not disjoint, hence the values are assigned by the first matching condition.

4.3 Capability Usage

The third category of features exploits the fact that some encoders do not implement all functions specified in the MP3 standard. We call this category *capability usage* and clearly separate these capabilities from surface parameters, such as header flags, because the latter can easily be changed without touching the compressed data.

4.3.1 Feature 7: SCFSI Usage

The *Scale Factor Selection Information* (SCFSI) is a parameter that allows an encoder to reuse scale factors for subsequent parts of the stream if they do not change over time. However, this compression method is only used by few encoders, namely `lame`, `gogo`, and `xingac21` ("AudioCatalyst"). We define a feature x_7 which reflects the use of SCFSI:

$$f_7(m) = \begin{cases} 0 & \text{for} \quad \mathrm{scfsi}(b_i) = 0 \quad \forall\, i\colon 1 \le i \le q \\ 1 & \text{else.} \end{cases}$$

4.3.2 Feature 8: Frame Length Alignment

Although MP3 frames have a fixed length, the amount of information used to describe the respective audio signal may vary. We refer to this quantity as *effective frame length* $\mathrm{len}_{\mathrm{eff}}(i)$. According to the MP3 standard, the effective frame length has no constraints to match a multiple of bytes, words or quad-words. However, we observed that some encoders (`8hz-mp3`, `bladeenc`, `m3ec`, `plugger`, `shine`, `soloh`) adjust all effective frame lengths to byte boundaries, while others do not. We use this characteristics as feature x_8:

$$f_8(m) = \begin{cases} 0 & \text{for} \quad \mathrm{len}_{\mathrm{eff}}(i) = 0 \bmod 8 \quad \forall\, i\colon 1 \le i \le r \\ 1 & \text{else.} \end{cases}$$

4.3.3 Feature 9: Huffman Table Selection

After the quantisation, the MDCT coefficients are further compressed by a Huffman style entropy coder. In contrast to the method proposed by Huffman [10], the tables are not computed from the marginal symbol distribution. In order to avoid the transmission of marginal distributions or table data, the developers of MP3 standardised a set of 28 pre-defined Huffman tables that were empirically optimised for the most probable cases in audio compression. In the very rare case of longer code words an escape mechanism allows storage of uncompressed values. An MP3 encoder chooses the most suitable table separately and independently for each of the three *regions* of the *big value* MDCT coefficients. As there is no efficient method to perform an optimal table selection, some encoders increase performance by using heuristics to quickly select a suitable table, rather than the optimal one. From a comparison of table usage frequencies, we found two noteworthy characteristics: First, all Xing encoders seem to avoid strictly using table number 0 for region 2.[8] Second, only a few encoders (`m3ec`, `mp3enc31`, `uzura`) use table 23 for the regions 1 and 2. We exploit these observations as additional information for our classification:

$$f_9(m) = \begin{cases} 0 & \text{for} \quad \text{table}_2(b_i) \neq 1 \quad \forall\, i\colon 1 \leq i \leq q \\ 1 & \text{for} \quad \exists (b_i, j)\colon\ \text{table}_j(b_i) = 23, \\ & \qquad\qquad 1 \leq i \leq q,\ j = 1, 2 \\ 2 & \text{else.} \end{cases}$$

Also, `shine` uses only a subset of the defined tables. However, as we can already identify this rarely used encoder with several other features, we refrain from adjusting this feature for the detection of `shine`.

4.4 Miscellaneous

4.4.1 Feature 10: Stuffing Byte Values

Since our last feature does not fit in any of the above categories, we decided to explain it separately. Independent from whether the reservoir mechanism is used or not, there may be a couple of bytes unused and filled up to meet the fixed frame length. These so-called *stuffing bits* can be set to any arbitrary values. For a closer examination of these values, we composed histograms of the byte values in the stuffing areas. While most encoders set all stuffing bits to zero, we still found some exceptions and mapped them into a symbolic feature x_{10}:

$$f_{10}(m) = \begin{cases} 0 & \text{for stuffing with zeros} \\ 1 & \text{for no stuffing at all} \\ 2 & \text{for stuffing with } \texttt{0x55} \text{ or } \texttt{0xaa} \\ 3 & \text{for stuffing with ``}\texttt{GOGO}\text{'' (}\texttt{0x47} \text{ and } \texttt{0x4f}\text{)} \\ 4 & \text{else.} \end{cases}$$

4.4.2 Concluding Remarks

The enumeration of features in this section is a subset of particularities we took into account and from which we selected the most promising ones. The selection is far comprehensive, so it is still feasible to find further differentiating features. Such features may be necessary to reliably separate new encoders, or encoders that were not included in our initial analysis.

[8]According to the standard, we count the regions from zero.

5 Experimental Results

For our experimental work, we used the R Statistical Framework [12, 23] and implemented an extension for statistical analyses of MP3 files on the basis of the open source MP3 player `mpg123` [9]. All results are based on an MP3 database of about 2,400 files encoded with 20 different encoders (cf. Table 5 in the Appendix A.1). The audio data was selected from different sources to make the measurements independent from specific types of music or speech. We included tracks from a re-mastered CD of Grammy Nominees, from a compilation of Blues Brothers (including some live recordings), further piano music by Chopin, as well as *Sound Quality Assessment Material* (SQAM) files with speech and instrumental sounds. All source files were read from CD recordings and stored as PCM wave files with 44.1 kHz, 16 bit, stereo.

If provided, we encoded every source audio with three constant bit rates that we believe are the most widely used rates for MP3 files, namely 112 kbps, 128 kbps, and 192 kbps. Additional MP3 files with variable bit rates, with two quality settings each, were generated by the encoders `iTunes`, `lame`, and `xingac21`.

5.1 Validity for Known Data

To measure the performance of our proposed method, we implemented a Naïve Bayes Classifier (NBC) [4] for fixed feature vectors of both symbolic and continuous features. In the first experiment, we trained the classifier c_{T_1} with a training set T_1 of about 2,400 cases. For each case, we extract a feature vector $\mathbf{f}(m_i)$ from a file encoded with a defined encoder e_i and use these tuples to induce classification parameters for c_{T_1}. To evaluate the performance of c_{T_1} we use the same feature vectors, because $S_1 = T_1$, as input to the classifier and compare the predicted encoders to the known true values. The results are shown in Table 6 in the Appendix A.1. In this experiment we reach a hit rate of $p(c_{T_1}, T_1) = 96.2\,\%$. As a measure of confidence, we calculate the average a-posteriori probability over the predicted encoders $\bar{P}_{\max} = 96.1\,\%$. The classifier calculates the a-posteriori probability $\max_i P(e_i|\mathbf{f}(m))$ for each file on the basis of the feature vector.

Table 4 summarises the features proposed in Section 4. We use a jack-knife method to empirically evaluate the importance of each feature for the classification result. Therefore the training and classification procedure is repeated several times, while excluding individual features one by one. The resulting increase of misses in the classification table is a measure for the importance of a feature. According to these values, the effective bit rate seems to be the most important feature, followed by the method of reservoir usage.

A closer look at the results shows that the main sources for classification errors occur between tightly related encoding engines, such as the DOS and UNIX versions of Fraunhofer's `l3enc`, and between two subsequent versions of Xing encoders (`xing3` and `xing99`). Also, `soloh` produces false classifications as `8hz-mp3`, especially for source files from the Blues Brothers CD. To explore these misclassifications, we debugged the `soloh` binary and found references to an early version of the `8hz-mp3` encoder. Hence, the similarity in statistical features may reveal insights about the "intellectual origin" of certain encoders.

To support our results and reduce the risk of tautological finding, we repeat the experiment with a split-half method. We trained the classifier c_{T_2} with a sub sample T_2 of the first training set T_1. All other elements from T_1 are used in the test set S_2, so that $T_2 \cap S_2 = \emptyset$. The results of this second experiment are shown in Table 7 (in the Appendix A.1). We found an overall hit rate of $p(c_{T_2}, S_2) = 94.9\,\%$ and an average a-posteriori probability of 95.9 %. As both quality measures differ only marginally from the first experiment (-1.3 and -0.2 percentage points, respectively), we conclude that the proposed method can also reliably identify the encoders of unknown MP3 files.

5.2 Reliability for Unknown Data

We used classifier c_{T_1} to determine the encoders of a random sample of 3,000 MP3 files drawn from different sources, for a total amount of more than 19,000 MP3 files. The overall average a-posteriori probability is 86.9 %. This is about 10 percentage points below the values for known data. We still consider this a good value because we are aware that our training set certainly does not include all available encoders. In addition, some well separable encoders in our assembled test database, such as `uzura` and `shine`, have not been identified in the mass of unknown files.

Figure 5 shows the distribution of a-posteriori probabilities for known and unkown test data.[9] The average a-posteriori probability for the most frequent encoders is shown separately in Table 3.

Encoders e_* not included in the training procedure may lead to misclassifications in either of two ways: If the feature vector $\mathbf{f}(m_*)$ is similar to one of the trained encoders then we face a misclassification without noticing it. In this case, the classifier reports a high a-posteriori probability, also interpreted as confidence measure, and the known features are "blind" towards the differences between the two encoders. To overcome this problem, one has to search for new features between the existing and the new encoders. In the second case, the feature vector $\mathbf{f}(m_*)$ is dissimilar from the typical values of the trained encoders. In this case, the classifier reports a low a-posteriori probability signifying the difficulties in assigning the actual feature vector to one of the trained classes. This case is more favourable because the classification problem is identified. The new encoder can be added to the classifier by retraining its parameters with an extended set.

[9]The latter has been reduced to bit rates between 112 kbps and 192 kbps, keeping 2912 files.

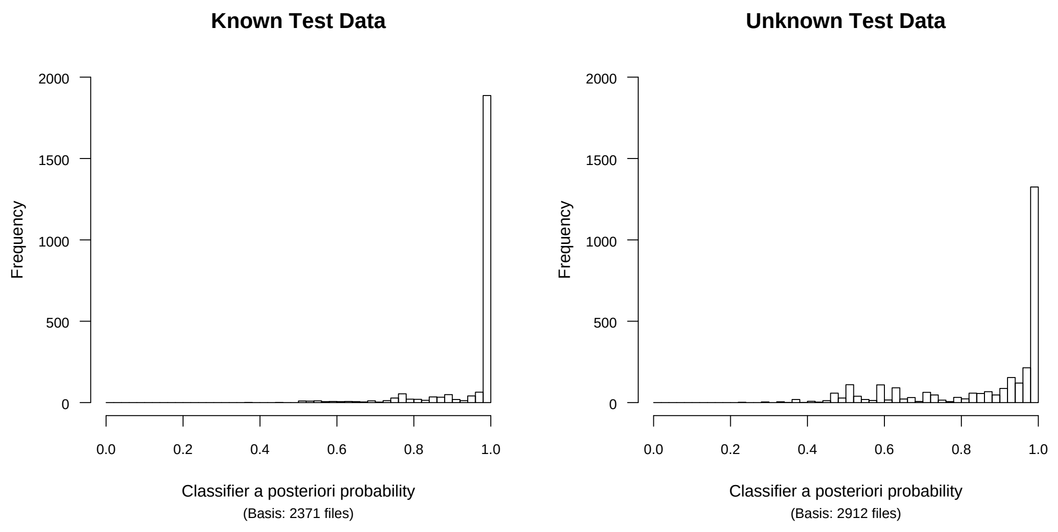


Figure 5: Comparison of classification confidence between self generated test data (left) and data collected in the wild (right).

5.3 Application for Steganalysis

To demonstrate the advances in steganalysis due to preclassification, we assembled a test set of about 500 pristine MP3 files from different encoders together with 369 steganograms from `MP3stego` [22]. If we run the attack against `MP3stego` [26] directly on the test set, we clearly identify all 369 steganograms but face an additional 377 false positives (75.4 %). Using the proposed method as preclassifier to filter all files from other encoders but `8hz-mp3` removes all false alarms, while still 312 steganograms are reliably detected. The miss rate of 15 % can further be reduced by using a specially trained classifier for this purpose. Only in combination with source classification does the detection method have sufficient discriminative power to be suitable for a large scale search for steganograms in MP3 files.

Table 3: A-posteriori probabilities for frequently identified encoders

			Confidence $P_{\max}(e\|\mathbf{x})$		
Rank	Encoder	Share	μ	σ	n
1.	`fhgprod`	17.3 %	0.87	0.14	381
2.	`xingac21`	11.6 %	0.99	0.04	256
3.	`l3enc272`	10.7 %	0.95	0.08	235
4.	`soundjam`	10.0 %	0.97	0.11	220
5.	`bladeenc`	9.0 %	0.83	0.16	198
6.	`mp3comp`	8.8 %	0.88	0.16	193
7.	`lame`[a]	8.6 %	0.56	0.11	190

[a]The low confidence may be due to different versions of `lame`; our training data has been encoded with the recent V 3.93.

6 Discussion and Conclusion

In this report, a method is presented to determine the encoder of ISO/MPEG 1 Audio Layer-3 data on the basis of statistical features extracted from the data. We explained a set of 10 features that were used with a *Naïve Bayes Classifier* to discriminate between 20 different MP3 encoders. The results show, that the proposed method is quite reliable for special purpose test data as well as for a sample of arbitrary MP3 files. However, the proposed scheme is far from being the ultimate solution and it needs further refinement for real world applications.

6.1 Limitations and Future Directions

The first obstacle is the relatively narrow range of supported bit rates. In order to keep the test database operable, we decided to concentrate on the most widely used bit rates. Moreover, we tried to keep the features independent from the bit rate. This approach appears to have been effective, as we do not have any problems when classifying variable bit rate (VBR) files despite never explicitly designing a feature for VBR data. However, as some encoders change the stereo model for different bit rates—especially for more extreme settings—further analyses of the robustness of the features against bit rate changes may increase the reliability of the classification.

As already stated, MP3 files support different stereo modes and most encoders offer a variety of options to fine tune the encoding result. Since the test database always uses the (most likely) default settings and the presented features do not care about other encoding modes, sophisticated encoding parameters may cause false classifications. Hence, the influence of stereo modes and other encoding options is subject to further research.

In addition, some of the present features rely on file parameters (e. g., total file size) or precisely evaluate the beginning of a track (e. g., the initial silence). These features will fail if only fragments of a stream shall be classified.

Regarding the composition of encoders in the training set, we mainly cover open source encoders and the most widely used encoders from Fraunhofer and Xing. The versions we researched were not systematically selected. Even if we are quite confident that additional software encoders can be added with moderate effort, we still have not examined the characteristics of hardware encoders which, for example, are used in portable digital audio recorders. The typical optimisations that are necessary to implement the MP3 encoding algorithm in DSP hardware might cause features of a different kind than those we exploit to differentiate software encoders.

To complete the list of open research questions, we refer to possible interactions between statistical features used for source classification and audio watermarking algorithms.

6.2 Transferability to Other Compression Formats

The results on MP3 files show that encoder detection is feasible and has useful applications for steganalysis and related areas. Hence, it might be an interesting question as to whether the approach can be generalised—certainly with adapted features—to other data formats.

Obviously, the MP3 format is a good candidate for encoder detection for two reasons: First, the popularity of the format, and thus the demand for encoders, developed a market for a couple of parallel developments in the late 1990s. Second, the inclusion of a psycho-acoustic model simplifies the task of feature discovery, because it leverages small numerical differences in the signal decomposition to measurable statistics, such as block type frequencies. From this point of view, MPEG 2 audio or MPEG 4 video seem to be promising formats for similar research. Other formats, for example the popular JPEG image compression scheme, might be quite harder to classify. This format is less complicated—at least in the way it is used in the overwhelming majority of cases—and the *Independent JPEG Group* (IJG) offers a standard implementation that is included in many applications [13].

However, judging from our experience with MP3, we are confident that similar methods can be constructed for most complex standards that leave latitude for implementations. Assuming that latitude increases with complexity, we can even be quite optimistic for future formats. Some discoveries we made, for example the block type signature of open source encoders, back our optimism: As long as programmers leave identifying traces by even violating the standards, whether unintentional or motivated for one's ego, classification will be feasible. Nevertheless, it is likely to always remain as an iterative analytical task, which is difficult to automate.

6.3 Related Applications

Apart from the advances in steganalytic reliability, the proposed method may have applications in two further ways. From an academic point of view, the insights gained from the analysis of inter-encoder differences in MP3 files can be used to construct new steganographic algorithms. If we know the parameters that are treated differently by different encoders, we can consider them as indeterministic and modify them to carry steganographic messages. Also, the design of watermarking algorithms, which are robust against MP3 compression, gains from further knowledge about encoder differences.

Last but not least, a more practical application for tools derived from this approach is digital forensics. Knowledge about the encoder of a suspicious file may lead to inferences about a possible creator. However, we must note that it is still possible to fool any of the presented features, at least if some effort is spent. The output of any of these classifiers is always a probabilistic guess and must not be considered as outright proof.

References

[1] 8hz-mp3 (V 02b), 8Hz Productions, `http://www.8hz.com/mp3/`, 1998.

[2] Brandenburg, K., Stoll, G.: ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio. *Journal of the Audio Engineering Society* 42 (10), 1994, pp. 780–794.

[3] DOSEMU (DOS Emulation) Main Page, `http://www.dosemu.org`.

[4] Duda, R. O., Hart, P. E.: Pattern Classification and Scene Analysis. Wiley, New York, 1973.

[5] EncSpot - An MP3 Analyzer, `http://www.guerillasoft.nstep.com/EncSpot2`.

[6] GNU Lesser Genreal Public License (LGPL). Version 2.1, `http://www.gnu.org/copyleft/lesser.html`, 1999.

[7] Fraunhofer Institut Integrierte Schaltungen (IIS), `http://www.iis.fraunhofer.de`.

[8] Fridrich, J.: Feature-based Steganalysis for JPEG Images and its Implications for Future Design of Steganographic Schemes. Paper to be presented at the Sixth Workshop on Information Hiding, Toronto, Canada, May 2004.

[9] Hipp, M.: MPG123 – Fast MP3 Player for Linux and UNIX Systems, `http://www.mpg123.de`, 2001.

[10] Huffman, D.: A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE* 40, 1962, pp. 1098–1101.

[11] IEC 958, Digital Audio Interface, International Standard, 1990.

[12] Ihaka, R., Gentlemen, R.: R – A Language for Data Analysis and Graphics. *Journal of Computational Graphics and Statistics* 5 (3), 1996, pp. 299–314.

[13] Independent JPEG Group, `http://www.ijg.org`.

[14] ISO/IEC 13818-3, Information Technology. Generic Coding of Moving Pictures and Associated Audio: Audio. International Standard, 1994.

[15] Nilsson, M.: ID3v2 – The Audience is Informed, `http://www.id3.org`, 1998.

[16] Langley, P., Iba, W., Thompson, K.: An Analysis of Bayesian Classifiers. *Proceedings of the 10th Conference on Artificial Intelligence*, MIT Press, 1992, pp. 223–228.

[17] Lindley, D. V.: Bayesian Statistics – A Review. Society for Industrial and Applied Mathematics, 1995.

[18] Lyu, S., Farid, H.: Detecting Hidden Messages Using Higher-Order Statistics and Support Vector Machines. In Petitcolas, F. A. P. (Ed.): Information Hiding. Fourth International Workshop, LNCS 2578, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 340–354.

[19] Moddemeijer, R.: On Estimation of Entropy and Mutual Information of Continuous Distributions. *Signal Processing* 16 (3), 1989, pp. 233–246.

[20] Pearl, L.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (2nd ed.). Morgan Kaufman, New York, 1992.

[21] Petitcolas, F. A. P., Anderson, R. J., Kuhn, M. G.: Information Hiding – A Survey. *Proceedings of the IEEE* 87 (7), 1999, pp. 1062–1078.

[22] Petitcolas, F. A. P.: MP3Stego, `http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego/`, 2002.

[23] The R Project for Statistical Computing, `http://www.r-project.org`.

[24] Stego-Lame, `http://sourceforge.net/projects/stego-lame/`.

[25] VMware is Virtual Infrastructure, `http://www.vmware.com`.

[26] Westfeld, A.: Detecting Low Embedding Rates. In F. A. P. Petitcolas (Ed.): Information Hiding. Fourth International Workshop, LNCS 2578, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 324-339.

[27] Wine HQ. An Open Source implementation of the Windows API on top of X and Unix, `http://www.winehq.com`.

A Appendix

A.1 Supplementary Tables

Table 4: Overview of features used for classification

No.	Description	Levels	Importance[a]
Size control features			
x_1	Effective bit rate ratio	4	8.35
x_2	Granule size balance	4	0.08
x_3	Reservoir usage ramp	3	5.01
x_4	Entropy of big values	cont.	2.15
Model decision features			
x_5	Preflag ratio	9	1.73
x_6	Block type transitions	5	1.56
Capability usage features			
x_7	SCFSI usage	2	0.50
x_8	Frame length alignment	2	0.92
x_9	Huffman table selection	3	0.63
Miscellaneous features			
x_{10}	Stuffing byte values	5	0.88

[a]The importance is measured with a jack-knife method: The column shows the additional overall classification error in percentage points if the feature is left out. Hence, higher values indicate higher importance of a feature.

Table 5: List of Analysed MP3 Encoders

Mnemonic	Name	Publisher	Version	Year
`8hz-mp3`	8HZ-MP3 Encoder	8Hz Productions	02b	1998
`bladeenc`	BladeEnc	Tord Jansson	0.94.2	2001
`fastenc`	FastEnc	Fraunhofer IIS	1.02	2000
`fhgprod`	Fraunhofer MP3 Producer	Opticom	2.1	1998
`gogo`	gogo301 petit	Herumi and Pen	3.01	2001
`iTunes`	Apple iTunes	Apple Computer Inc.	4.1-52	2003
`l3enc272`	l3enc (Linux)	Fraunhofer IIS	2.72	1997
`l3encdos`	l3enc (MS-DOS)	Fraunhofer IIS	2.60	1996
`lame`	LAME Ain't an MP3 Encoder	Mike Cheng et al.	3.93	2003
`m3ec`	M3E Command Line Version	N/A	0.98b	2000
`mp3comp`	MP3 Compressor	MP3hC	0.9f	1997
`mp3enc31`	mp3enc (Demo)	Fraunhofer IIS	3.1	1998
`plugger`	Plugger	Alberto Demichelis	0.4	1998
`shine`	Shine	Gabriel Bouvigne	0.1.4	2001
`soloh`	SoloH MPEG Encoder	N/A	0.07a	1998
`soundjam`	SoundJam (Macintosh)	Casady and Greene	2.5.1	2000
`uzura`	Uzura 3	N/A (Fortran code)	1.0	2002
`xing3`	Xing MP3 Encoder	Xing Technology Corp.	3.0-32	1997
`xing98`	Xing MP3 Encoder (x3enc)	Xing Technology Corp.	1.02	1998
`xingac21`	AudioCatalyst	Xing Technology Corp.	2.10	1999

Note: All trademarks are the property of their respective owners.

Table 6: Classifier performance on training data ($S = T$)

% of files classified as …	8hz-mp3	bladeenc	fastencc	fhgprod	gogo	iTunes	l3enc272	l3encdos	lame	m3ec	mp3comp	mp3enc31	plugger	shine	soloh	soundjam	uzura	xing3	xing98	xingac21
	True Encoder																			
8hz-mp3	100	–	–	–	–	–	–	–	–	–	–	–	–	–	21	–	–	–	–	–
bladeenc	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
fastencc	–	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
fhgprod	–	–	–	94	–	–	–	–	–	–	20	–	–	–	–	–	–	–	–	–
gogo	–	–	–	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
iTunes	–	–	–	–	–	100	–	–	–	–	–	2	–	–	–	–	–	–	–	–
l3enc272	–	–	–	–	–	–	85	9	–	–	–	–	–	–	–	–	–	–	–	–
l3encdos	–	–	–	–	–	–	15	91	–	–	–	–	–	–	–	–	–	–	–	–
lame	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–	–	–	–	–	–
m3ec	–	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–	–	–	–	–
mp3comp	–	–	–	6	–	–	–	–	–	–	80	–	–	–	–	–	–	–	–	–
mp3enc31	–	–	–	–	–	–	–	–	–	–	–	98	–	–	–	–	–	–	–	–
plugger	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–	–
shine	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–
soloh	–	–	–	–	–	–	–	–	–	–	–	–	–	–	79	–	–	–	–	–
soundjam	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–
uzura	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–
xing3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	89	10	–
xing98	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	11	90	–
xingac21	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100
	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

Basis: $|T| = |S| \approx 2400$ files, $n = 20$ encoders, $k = 10$ features, overall error rate 3.8 %, average classification confidence $\bar{P}_{\max} = 0.961$

Table 7: Classifier performance on disjoint test data ($S \cap T = \emptyset$)

	True Encoder																			
	8hz-mp3	bladeenc	fastencc	fhgprod	gogo	iTunes	l3enc272	l3encdos	lame	m3ec	mp3comp	mp3enc31	plugger	shine	soloh	soundjam	uzura	xing3	xing98	xingac21
% of files classified as …																				
8hz-mp3	95	–	–	–	–	–	–	–	–	–	–	–	–	–	23	–	–	–	–	–
bladeenc	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
fastencc	–	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
fhgprod	–	–	–	94	–	–	–	–	–	–	38	–	–	–	–	–	–	–	–	–
gogo	–	–	–	–	100	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
iTunes	–	–	–	–	–	100	–	–	–	–	–	2	–	–	–	–	–	–	–	–
l3enc272	–	–	–	–	–	–	84	–	–	–	–	–	–	–	–	–	–	–	–	–
l3encdos	–	–	–	–	–	–	16	100	–	–	–	–	–	–	–	–	–	–	–	–
lame	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–	–	–	–	–	–
m3ec	–	–	–	–	–	–	–	–	–	100	–	–	–	–	3	–	–	–	–	–
mp3comp	–	–	–	6	–	–	–	–	–	–	62	–	–	–	–	–	–	–	–	–
mp3enc31	–	–	–	–	–	–	–	–	–	–	–	95	–	–	–	–	–	–	–	–
plugger	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–	–
shine	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–	–	–
soloh	5	–	–	–	–	–	–	–	–	–	–	–	–	–	74	–	–	–	–	–
soundjam	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–	–
uzura	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100	–	–	–
xing3	–	–	–	–	–	–	–	–	–	–	–	3	–	–	–	–	–	85	13	–
xing98	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	15	87	–
xingac21	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	100
	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

Basis: $|T| = |S| \approx 1200$ files, $n = 20$ encoders, $k = 10$ features, overall error rate: 5.1 %, average classification confidence $\bar{P}_{\max} = 0.959$

A.2 Sample Session

This section shows how the analytical process under R can be handled and presents an example for encoder classification. To follow the session the following software is required:

1. R Statistical Framework, free download under `http://www.r-project.org`
2. The `mp3.zip` (Windows binary) or `mp3.tar.gz` (Unix source) package developed for CHAMP3
3. The feature extraction and classification code (file name: `champ3.R`)
4. A training set of feature vectors (file name: `feature.list.RData`)
5. Sample MP3 files from recommended encoders

After R is successfully set up, the CHAMP3 import library has to be installed into the R framework. On windows, start the R GUI environment and select "*Install package(s) from local zip file ...*" in the "*Packages*" menu. Then choose the location of `mp3.zip` and confirm. On Unix systems, this is accomplished by the shell command line

```
bash# R CMD INSTALL mp3
```

where the package `mp3.tar.gz` has to be in the working directory. Note, that super-user privileges may be required for this step.

Now switch into the R command line environment and call the library with

```
> library(mp3)
```

If everything is loaded correctly, you get the following output:

```
MP3 import library for R (based on mpglib)
(c) 2003-04 Technische Universität Dresden
URL: http://www.inf.tu-dresden.de/champ3
-------------------------------------------------
The development of this package was supported by
the Air Force Office of Scientific Research under
the research grant number FA8655-03-1-3A46.
-------------------------------------------------
```

To load an MP3 file and store it into an R object named `m`, type:

```
> m <- load.mp3("lame_01.mp3")
Start decoding ... decoding successful.
```

A help page explaining more details of the parameters of `load.mp3` is available by

```
> ?load.mp3
```

To display the structure of the MP3 object, just enter

```
> m
List of 6
 $ filename  : chr "lame_01.mp3"
 $ length    : int -1
 $ id3.length: int 0
 $ aid.length: int 0
 $ junk.bytes: int 0
 $ stuff.hist: int [1:255] 247 1 1 1 1 1 0 1 0 0 ...
'data.frame':   10268 obs. of  8 variables:
 $ length    : int  288 288 288 288 288 288 288 288 288 288 ...
 $ size      : int  417 417 418 418 418 418 418 418 418 418 ...
 $ header    : int  -290716 -290716 -290236 -290236 -290236 -290236 ...
 $ bitrate   : int  128 128 128 128 128 128 128 128 128 128 ...
 $ freq      : int  44100 44100 44100 44100 44100 44100 44100 44100 ...
 $ junk.bytes: int  0 0 0 0 0 0 0 0 0 0 ...
 $ backstep  : int  0 0 381 511 511 511 511 511 511 511 ...
 $ error     : logi  FALSE FALSE FALSE FALSE FALSE FALSE FALSE ...
'data.frame':   41072 obs. of  20 variables:
 $ frame.parent  : int  1 1 1 1 2 2 2 2 3 3 ...
 $ scfsi         : int  -1 -1 0 0 -1 -1 0 0 -1 -1 ...
 $ part23.length : int  0 0 0 0 0 0 0 0 0 0 ...
 $ big.values    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ stereo        : int  0 1 0 3 0 5 0 7 0 9 ...
 $ gain.0        : num  1 1 NA 1 1 1 1 1 1 1 ...
 $ gain.1        : num  1 1 1 1 1 1 1 1 1 1 ...
 $ gain.2        : num  1 1 1 1 1 1 1 1 1 1 ...
 $ gain.pow2     : num  1.11e-16 1.11e-16 1.11e-16 1.11e-16 7.07e-01 ...
 $ table.0       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ table.1       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ table.2       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ table.count1  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ region.1      : int  2 2 2 2 18 18 18 18 2 2 ...
 $ region.2      : int  4 4 4 4 288 288 288 288 4 4 ...
 $ block.type    : int  0 0 0 0 2 2 3 3 0 0 ...
 $ mixed.flag    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ compress.scale: int  0 0 0 0 0 0 0 0 0 0 ...
 $ scale         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ preflag       : int  0 0 0 0 0 0 0 0 0 0 ...
```

Single elements of this structure can be accessed by name. For example, to calculate the mean effective frame length in bits, enter

```
> mean(m$frames$length)
[1] 3317.785
```

Obviously, the whole set of statistical functions in R can easily be applied to MP3 data. To extract a feature vector, first load the CHAMP3 analysis functions into the workspace

```
> source("champ3.R")
```

and then use the `features` function on `m`:

```
> features(m)
               ENC            F1                F2            F3
         "unknown"    "bps_low"    "gran_bias_3"   "back_soft"
                F4            F5                F6            F7
"3.32592000290358"      "pre_2"     "block_lame"   "scfsi_yes"
                F8            F9               F10
             "bit"    "default"      "stuff_alt"
```

The way the features are extracted and the meaning of the symbolic values are explained below in Section 4. To assemble a training set, one would have to create a file list as input to the function `extract.features`. However, to facilitate this time-consuming step, we provide a pre-extracted feature list from our test database. The following command loads the feature list into the workspace:

```
> load("feature.list.RData")
```

Check the structure of `feature.list` with the function `str`:

```
> str(feature.list)
‘data.frame’:   2371 obs. of  11 variables:
 $ ENC: Factor w/ 20 levels "8hz-mp3","blade..",..: 1 1 1 1 1 1 1 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F1 : Factor w/ 4 levels "bps_0","bps_1",..: 3 3 3 3 3 3 3 3 3 3 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F2 : Factor w/ 4 levels "gran_bias_1",..: 2 2 2 2 2 2 2 2 2 2 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F3 : Factor w/ 3 levels "back_hard","bac..",..: 3 3 3 3 3 3 3 3 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F4 : num  3.53 3.53 3.56 3.63 3.56 ...
 $ F5 : Factor w/ 9 levels "pre_1","pre_2",..: 9 8 9 9 8 9 9 9 9 9 ...
```

```
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F6 : Factor w/ 5 levels "block_gogo","bl..",..: 5 5 5 5 5 5 5 5 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F7 : Factor w/ 2 levels "scfsi_no","scfs..": 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F8 : Factor w/ 2 levels "bit","byte": 2 2 2 2 2 2 2 2 2 2 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F9 : Factor w/ 3 levels "default","no-ta..",..: 1 2 1 1 1 1 1 1 1 2 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
 $ F10: Factor w/ 5 levels "stuff_alt","stu..",..: 1 3 3 1 1 3 3 1 5 1 ...
  ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
```

To induce a classifier from the training set, use the `nbc` (*Naïve Bayes Classifier*) function. As the part in the command line indicates, the encoder index `ENC` (e_i in mathematical notation) shall be predicted by all other components of the data frame `feature.list`.

```
classifier <- nbc(ENC~F1+F2+F3+F4+F5+F6+F7+F8+F9+F10,feature.list)
```

—or, much more briefly—

```
classifier <- nbc(ENC~.,feature.list)
```

The function `predict.mp3info` (short cut: `predict`) can be used to determine the most probable encoder of `m`. To access the first hit, type

```
> predict(m,classifier)$ENC[1]
[1] "lame"
```

Use `predict.files` to conveniently access the encoder classification for a given file name, or a list of names. The output of `predict.files` displays a ranking of the three most probable encoders together with the respective a-posteriori probabilities.

```
> predict.files("mp3/lame/grammynom98/audio_01.mp3",classifier)
[1] "mp3/lame/grammynom98/audio_01.mp3"
Start decoding ... decoding successful.
  rank      ENC      p.value bitrate
1    1     lame 1.000000e-00     128
2    2     gogo 5.162796e-10     128
3    3 xingac21 1.114803e-12     128
```

The source code `champ3.R` is also documented and contains further hints on the usage of the provided commands. It also includes an interface for a JSP-based web application for MP3 file analysis, which is online under `http://www.inf.tu-dresden.de/champ3`.